Machine Mosaic User Guide

Startup

STEP 1. ensure you are connected to "bricklaying network" (pw: csaildrl) and that the unitree G1 is on. This is a subnet of "statacenter", so you will still have internet connection

STEP 2. ssh with `$ssh unitree@192.168.123.165`

Troubleshooting:
if it has difficulty with "connecting to host", try it a couple more times, waiting to ensure the G1 is fully booted up. if the G1 has been running for a while and you still can't connect, it is probably that the G1 has crashed or booted improperly --- restart the G1 by power cycling via the power button on the battery

Ignore the ros prompt and any errors... just type in any number to begin

OPTIONAL STEP 3. (optional: see ubuntu desktop) to activate vnc for desktop viewing, run $x11vnc -forver -create. You can then connect to the desktop with remmina or any other vnc client --- type in 192.168.123.165 on your vnc client

once you have connected to vnc, type $startxfce4 in the terminal to start the xfce4 desktop manager

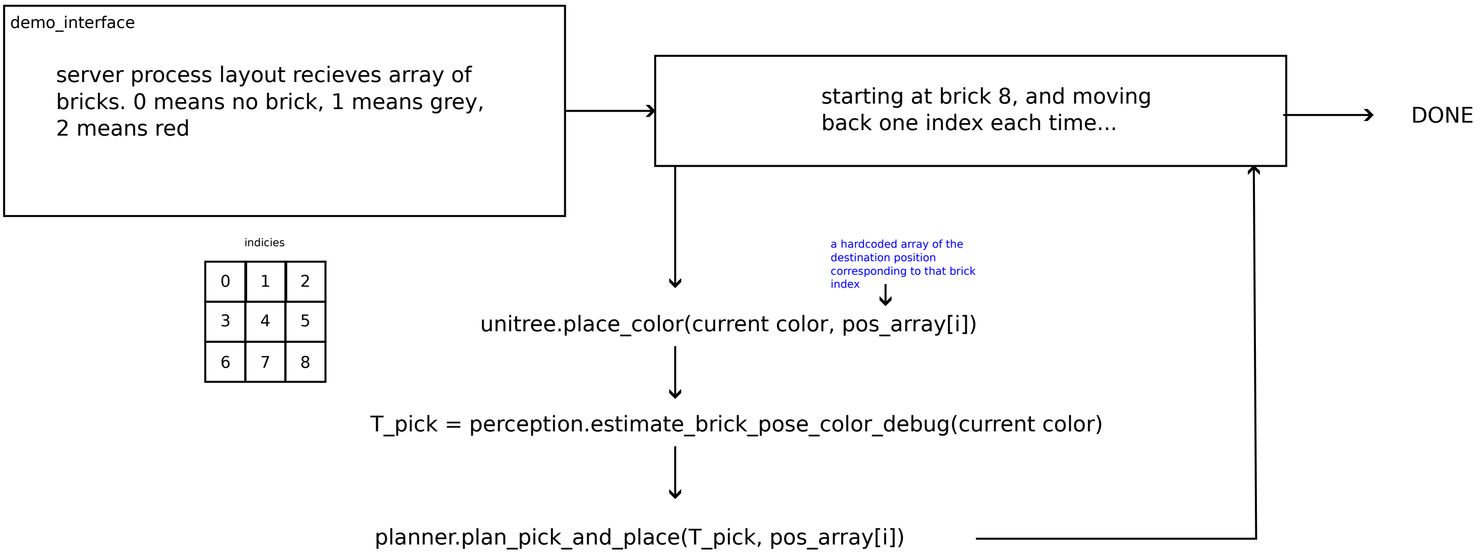
STEP 4. the repository is located in ~/drl/. Before running scripts, activate the conda environment with $conda activate unitree

TROUBLESHOOTING:

Terminal freezes / entire desktop freezes /ssh won't connect: the G1 is probably crashed. This happens every half-hour or so. We don't know why. Power cycling fixes the G1

"Can't find DEX3 hand controller": something similar occasionally pops up when running scripts. The hands did not connect properly upon boot. Power cycle as many times as it takes to resolve this issue. Spreading the fingers out before turning the unitree on improves probability of connecting properly

Program Flow: demo_interface.py



demo_interface

server process layout recieves array of bricks. 0 means no brick, 1 means grey, 2 means red

starting at brick 8, and moving back one index each time...

DONE

indicies

0	1	2
3	4	5
6	7	8

a hardcoded array of the destination position corresponding to that brick index

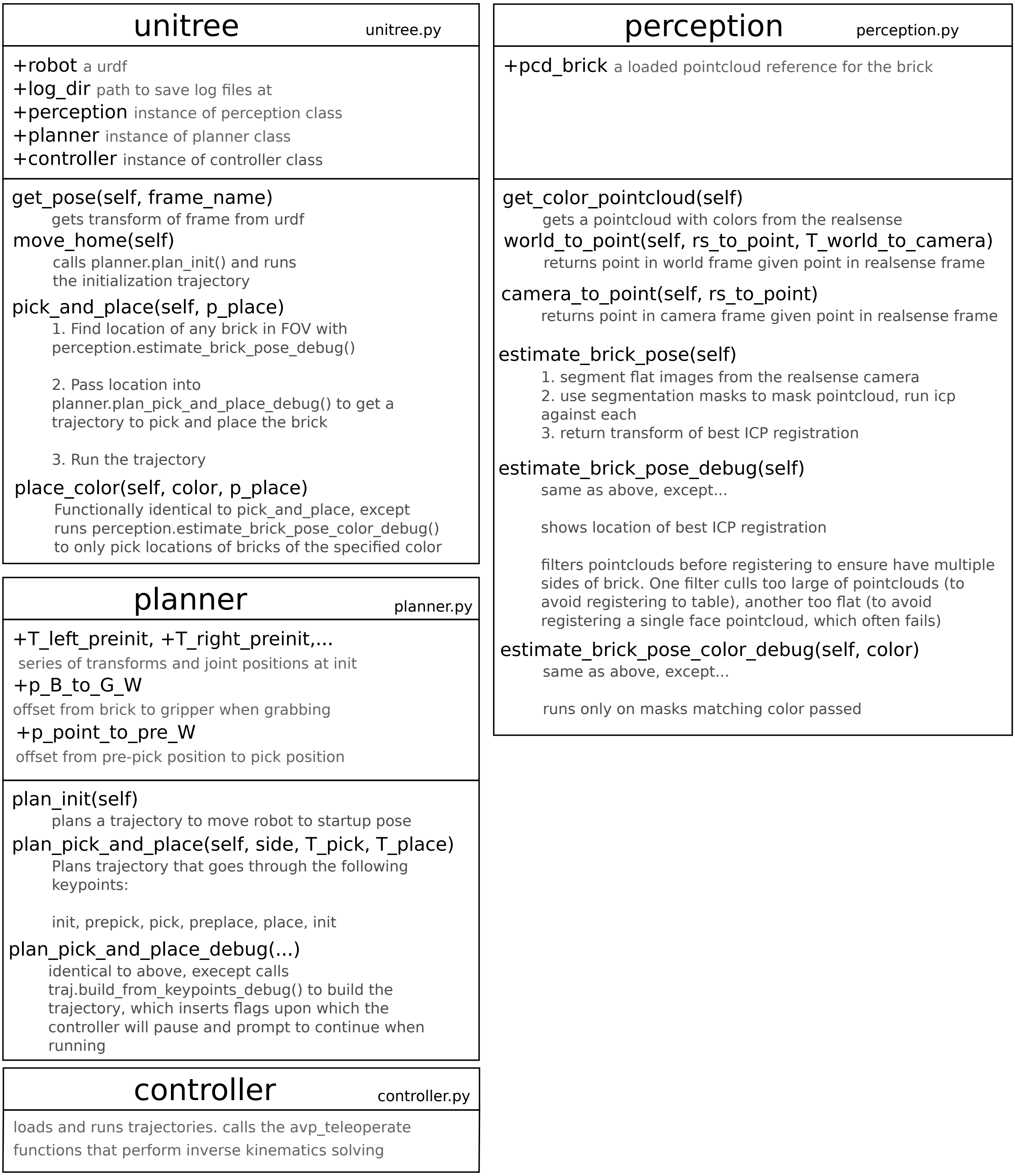
unitree.place_color(current color, pos_array[i])

T_pick = perception.estimate_brick_pose_color_debug(current color)

planner.plan_pick_and_place(T_pick, pos_array[i])

Structure

- /scripts/
  - demo_interface.py
  - pick_and_place.py
- /src/
  - controller.py
  - perception.py
  - planner.py
  - unitree.py



unitree

unitree.py

+robot a urdf
+log_dir path to save log files at
+perception instance of perception class
+planner instance of planner class
+controller instance of controller class

get_pose(self, frame_name)
gets transform of frame from urdf

move_home(self)
calls planner.plan_init() and runs the initialization trajectory

pick_and_place(self, p_place)
1. Find location of any brick in FOV with perception.estimate_brick_pose_debug()

2. Pass location into planner.plan_pick_and_place_debug() to get a trajectory to pick and place the brick

3. Run the trajectory

place_color(self, color, p_place)
Functionally identical to pick_and_place, except runs perception.estimate_brick_pose_color_debug() to only pick locations of bricks of the specified color

planner

planner.py

+T_left_preinit, +T_right_preinit,...
series of transforms and joint positions at init

+p_B_to_G_W
offset from brick to gripper when grabbing

+p_point_to_pre_W
offset from pre-pick position to pick position

plan_init(self)
plans a trajectory to move robot to startup pose

plan_pick_and_place(self, side, T_pick, T_place)
Plans trajectory that goes through the following keypoints:

init, prepick, pick, preplace, place, init

plan_pick_and_place_debug(...)
identical to above, execept calls traj.build_from_keypoints_debug() to build the trajectory, which inserts flags upon which the controller will pause and prompt to continue when running

controller

controller.py

loads and runs trajectories. calls the avp_teleoperate functions that perform inverse kinematics solving

perception

perception.py

+pcd_brick a loaded pointcloud reference for the brick

get_color_pointcloud(self)
gets a pointcloud with colors from the realsense

world_to_point(self, rs_to_point, T_world_to_camera)
returns point in world frame given point in realsense frame

camera_to_point(self, rs_to_point)
returns point in camera frame given point in realsense frame

estimate_brick_pose(self)
1. segment flat images from the realsense camera
2. use segmentation masks to mask pointcloud, run icp against each
3. return transform of best ICP registration

estimate_brick_pose_debug(self)
same as above, except...

shows location of best ICP registration

filters pointclouds before registering to ensure have multiple sides of brick. One filter culls too large of pointclouds (to avoid registering to table), another too flat (to avoid registering a single face pointcloud, which often fails)

estimate_brick_pose_color_debug(self, color)
same as above, except...

runs only on masks matching color passed